Old Lessons and New Challenges for Future Heterogeneous Systems

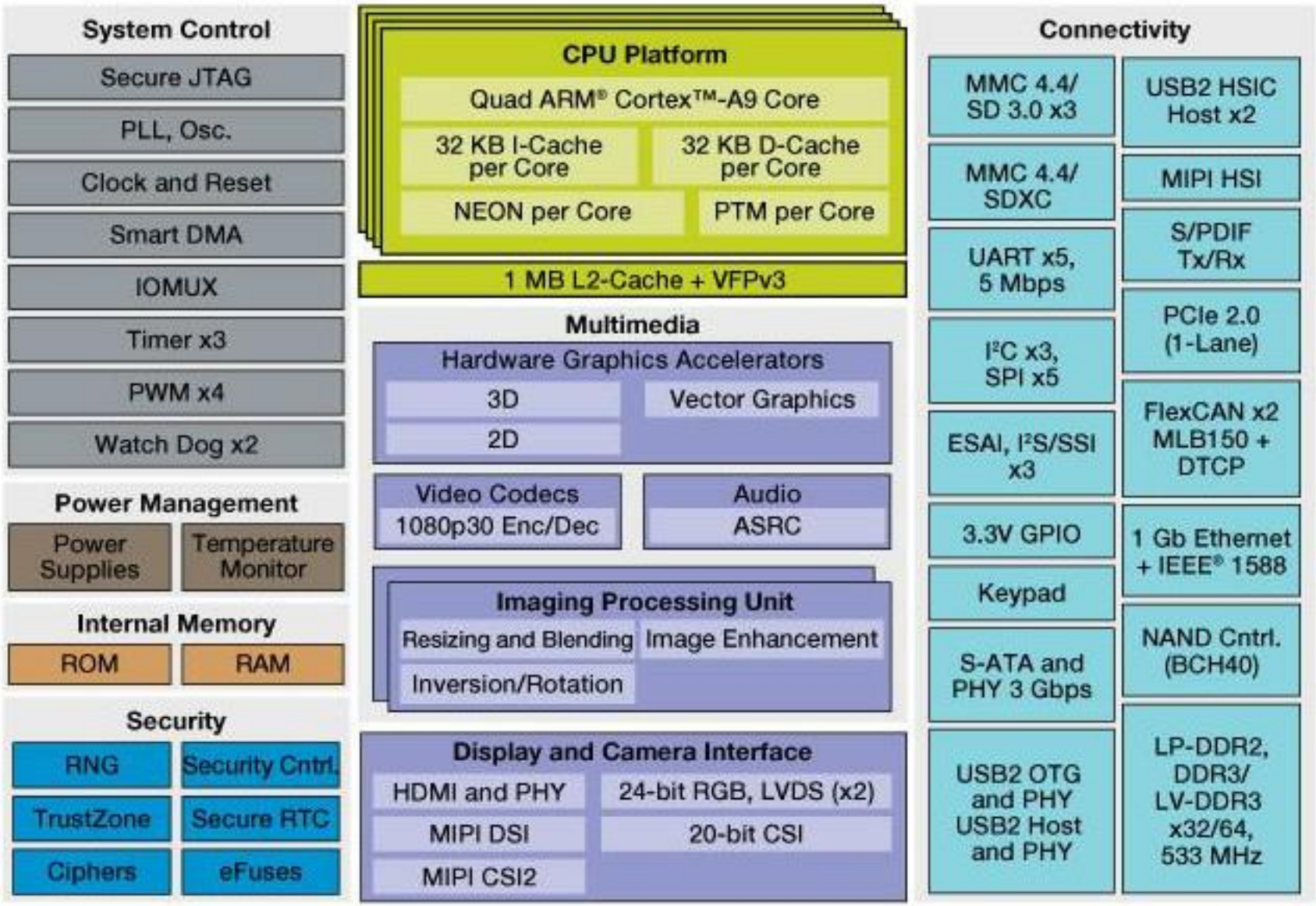
Bob Colwell

January, 2018

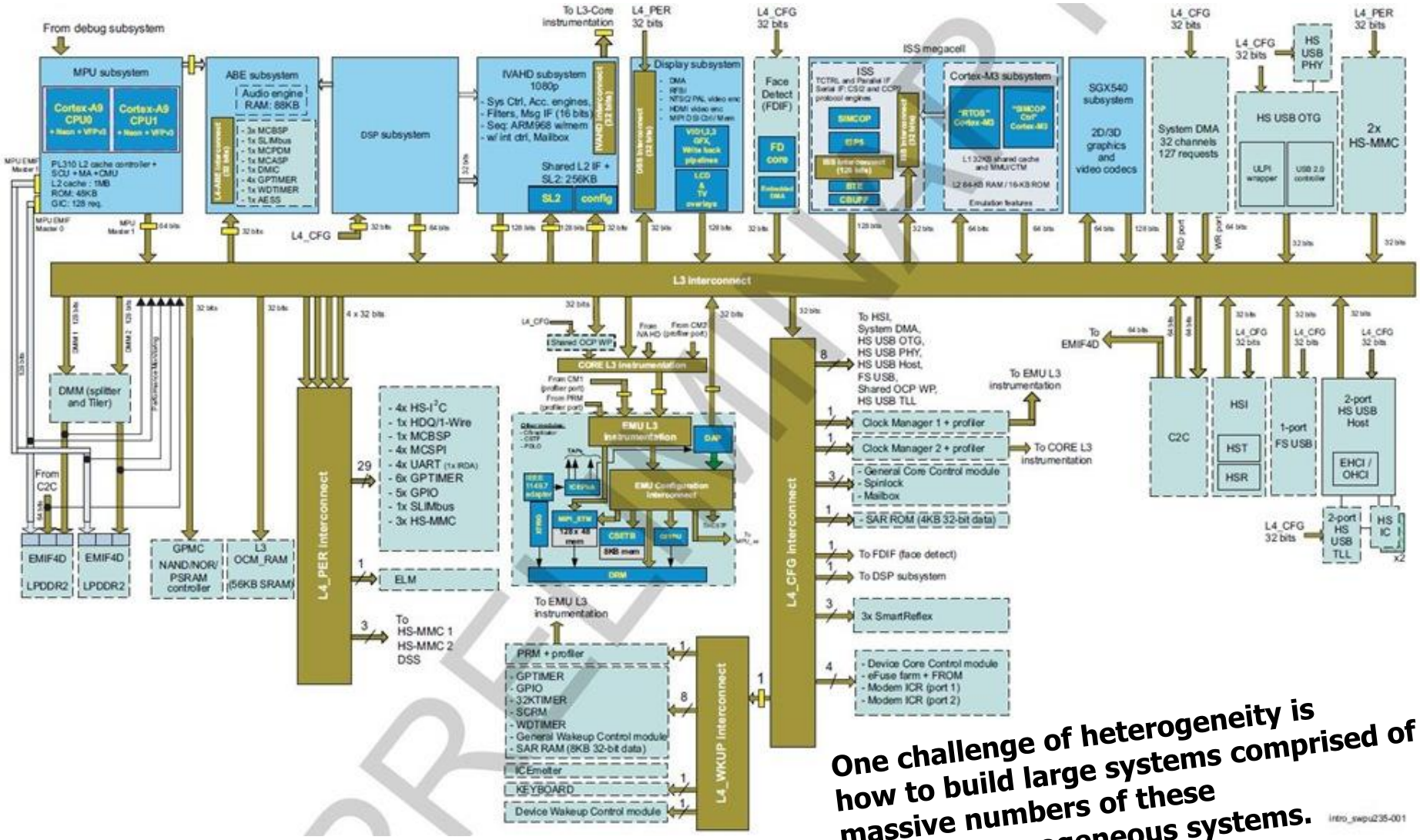
Extreme Heterogeneity Workshop

Gaithersburg, MD

Heterogeneity is already here: smartphone SoC ca 2016



Another example: TI OMAP



One challenge of heterogeneity is how to build large systems comprised of massive numbers of these already-heterogeneous systems.

What does our homogeneous past teach us?

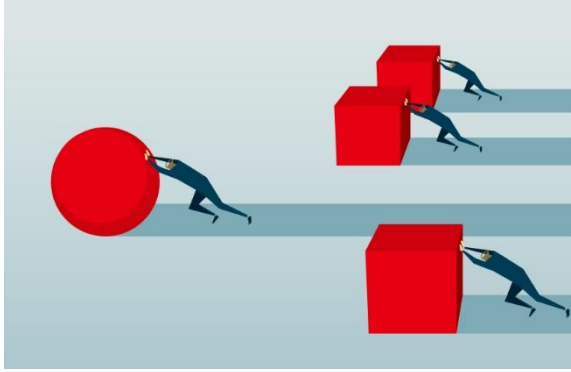
General-purpose computing ruled for decades because

- **Better perf on existing code + new apps & OS's = $$Profits**
  - (All else being equal.) Rinse and repeat.

- Why did that formula work?
  1. CPU perf/features got exponentially better over time
  2. OS's/SW got better (and demanded better CPU's)
  3. Platform improvements did not choke (PCI, QPI, USB, DRAM, buses, caches…)
  4. Overall system cost fell drastically
  5. Security issues have remained annoyances, not limiters
  6. There was a predictable future safeguarding today's investments

- Can we just re-apply that formula? No. ML is dead. But we must address a major limiter that was choking it anyway: efficiency.

What about efficiency?



General computing largely ignored efficiency

- MPEG-2 HW decoder 1000x > CPU SW decoder
- Let common tasks be provided in HW (that's heterogeneity)
- And tasks that are too much for CPU's may become feasible with accelerators (GPU's and beyond)
  - Don't forget, end of ML means you can no longer just wait around and faster machines will appear…only accelerators will enable certain class of new apps
  - What new apps? Dunno. Historically, we *never* knew until they appeared
    - It would be foolish to assume that won't happen again
    - But now accelerator designers may have to predict these new apps

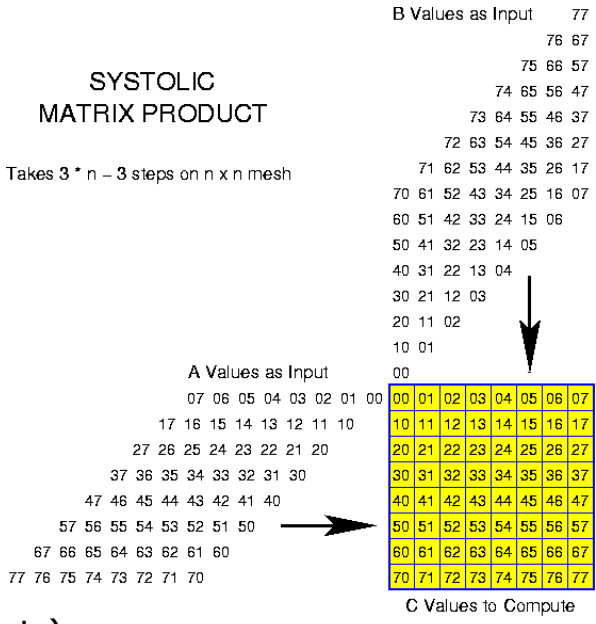
- 1000x was don't-care when CPU power low
- Efficiency became 1st order concern in 2004 when sys thermals hit air-cooling limit
  - Industry's answer: multicore
  - Kept the arch franchise going but w/o the customary perf kick

Clearly, multicore isn't the answer to the end of Moore's Law when each of those cores suffers from an analogous efficiency loss

The Future's Prime Directive

Thou Shalt Not Move Bits Around

- Hetero specialized accelerators help greatly
- Any prospects for "general purpose" accelerators?
  - Systolic arrays
  - Tiled approaches like Ambric's, Tilera's
  - Which raise research questions
    - Local caches to local interconnect BW ratios
    - Types and amounts of CPU performance vs avail mem
    - Amounts of instruction cache per tile
    - What apps are representative (enough to guide design targets)
    - What should be the programming model
    - And does that programming model need to coordinate with overall sys prog model?
    - Can/should we stream instructions as well as data across the fabric?



There are "easy" communication paths…

Get the comms paths right

- Get the intended ones right (BW, protocol, function, perf)
  - E.g., a dedicated pipeline where one unit feeds another directly
  - We're used to that kind of speeds-and-feeds design
  - Big/Little cores heterogeneity has also been tried
    - With varying success
    - Can't always tell which cores will be fastest on a given workload! Work needed here.
  - Industry now experimenting with FPGA's in system or on-die
    - Range of hetero behavior will be very wide
    - Comms FPGA/CPU? Shared memory? Shared caches?
  - Need more IEEE standard SoC interconnects
    - Leverage IEEE std design flow (IP-XACT?)

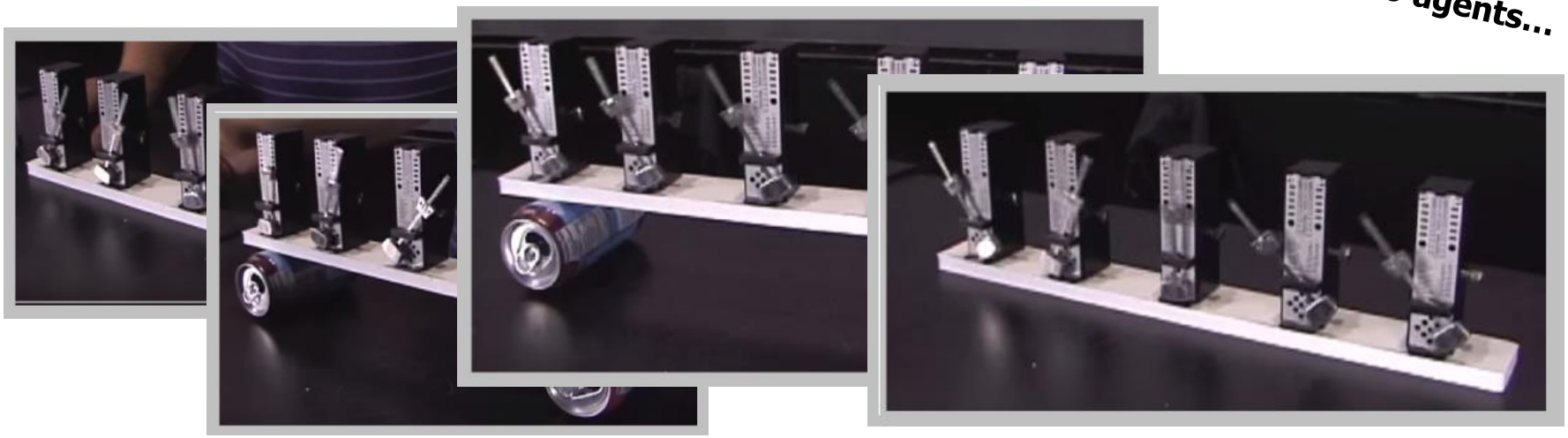
And there are "hard" communications paths

Manage the unintended comms paths properly

- Also known as "sneak" paths or "back" channels
- Any physical means by which conceptually distinct machines can (and therefore *will*) interact

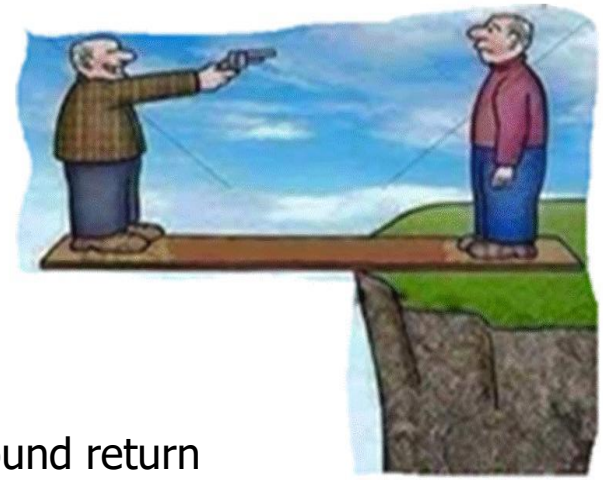
- Example 1: electrical ground bounce
  - Inductive voltage spikes on Vcc/gnd from fast signal edges
- Example 2: Spectre/Meltdown
  - Couples performance tweak to security hole
- Example 3: oscillator coupling

Think of these oscillators as async hetero agents…



We must take all comms paths into account

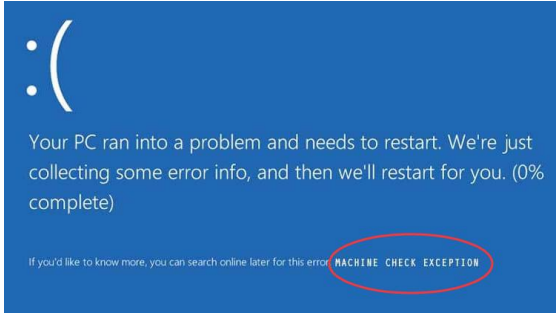
- "Tragedy of the Commons" paths (aka shared resources)
  - Power supply, ground returns, EMI
  - Thermals
  - Security-related behavior
  - Manage these despite inevitable design errata
    - Can't even assume such errata has no common mode!



- Thermals is the one I worry about most
  - Each hetero agent uses supply current, generates ground return current, and contributes to overall thermal load
  - SoC constitutes a closed-loop control system
    - Workload-related activity causes various hetero agents to heat up SoC
    - Variable cooling system drives temp back down
    - **How to manage each agent & whole system to best perf?**
    - **How to manage to any *guaranteed* minimum performance?**
    - How to prove whole thing is stable (no "poles in the right half plane")

Remember the coupling between temperature and O-ring elasticity

What about "machine check?"



After 40 years we have no standards in this area

- We don't systematically check machine ops nor results for correctness
- We don't have uniform means of constraining errors from propagating once they manifest…debug is hard and will get much harder

The **heterogeneous future is inheriting an ad hoc, crazy quilt** of

a. What's easy to measure (parity on ROMs, illegal FSM states, protocol errors)
b. Necessary to monitor (DRAM errors, cache errors, bus xfers)
c. Program errors (FP exceptions, illegal accesses)
d. Temperature & Volts (over/under)

What do we really want?

- Confirmation of correct answers?
- Trending towards marginality indicator to stimulate preventive maintenance?
- Health-of-the-machine indicator?
- AutoRecovery from certain errors? AutoRecovery from *all* errors? (good luck!)

Small % error likelihood x large number of trials = Big Problem

- Neither science nor engineering has really been applied here.

Our machine check past is not good enough for hetero.

Standards are not easy

Industry standards for hetero design will be crucial for hitting short time-to-market for all future systems.

Other Hetero Worries

- Issues associated with simult. heterogeneity at multiple levels
  - Individual chips managing their own thermals, voltages, sleep conditions in a large system which is doing similar things at higher levels
  - **Reproducibility and deterministic behavior are both at risk here**

- Implications of implementation tech now reaches back to algorithms and runtime environment
  - "Cheetah" algorithms that run fast but must stop to cool off may lose to tortoises
  - Full-up system emulation/simulation including thermals will be only way to intelligently make these tradeoffs at design time
    - Can we move some of them to runtime?

And in conclusion…

Carry forward lessons from past 40 years of (mostly) homogeneous systems while focusing on new challenges from hetero:

I. Get the intended comms paths right

II. Identify and explicitly manage unintended comms paths

III. See if "general purpose hetero engines" make sense

IV. Invest in machine check architectures

V. Get the standards right…we'll need them

VI. Remember where the profits come from